# FKM5 - User Manual

limes datentechnik gmbh

Version 5.1.20, May 10 2019

# Table of Contents

released

Frankenstein Limes Key Management Extension for version 5 (FKM5)



## Trademarks

Below, you can find all trademarks or registered trademarks of limes datentechnik ® gmbh. These trademarked terms are marked with the appropriate symbol (® or ™), indicating registered or common law trademarks owned by limes datentechnik ® gmbh at the time this information was published. The following terms are trademarks of limes datentechnik ® gmbh in Germany, other countries, or both:

- limes® - Short company name of the owner of this document
- limes datentechnik® - Company name of the owner of this document
- FLCL® - Frankenstein Limes Command Line
- FLCC® - Frankenstein Limes Control Center
- FLAM® - Frankenstein Limes Access Method
- FLUC® - Frankenstein Limes Universal Converter
- FLIES® - Frankenstein Limes Integrated Extended Security
- FLAMFILE® - A file based on FLAM syntax

## Abstract

libfkm5 joins all FKM5 implementations of limes datentechnik into one library. For each supported specification (FINPIN, KMIP, PGP) several implementations (Software (SWE), PKCS#11 (P11), IBM CCA) are available. This document describes all the currently available FKM5 functions.

The FLAM Key Management Extension in version 5 (FKM5) is a service provider interface to integrate FLAM5 into different cryptographic infrastructures.

# PREFACE

The FKM5 parameter list is parsed based on the CLE/P library. The CLE/P library was developed by limes datentechnik and released as open source under the ZLIB license. CLE/P is a compiler to provide a platform-independent command line interface for each kind of batch processing environment.

The CLE/P library provides a lot of features, including help and documentation. For example: We use this library to automatically create this document as part of our build process. If we add a parameter to the CLE/P tables and build the FL5 project, this manual will be regenerated as well in order to be always up to date.

This manual is generated with the INFO command of FLCL provided by CLE/P library by calling the command below:

```
flcl info get.fkm5(docu) output=fkm5book.txt
```

# Chapter 1. FKM5 overview

The FLAM key-management extension in version 5 (FKM5) links between FLAM's cryptographic protection mechanisms (privacy, integrity, completeness), various cryptographic infrastructures (PGP, KMIP, x509, FINPIN, ...), and various architectures of hardware security modules (HSM (IBM-CCA/ICSF, PKCS#11, ...)) or software implementations (PGP key rings) in order to provide a professional key management where access to data can be controlled. Of course, FLAM also supports protection by a simple passphrase or an internal constant as keys, but professional solutions are implemented by means of this service provider interface which has been available since FLAM version 5.

### Benefits

- Use of existing cryptographic infrastructures for protecting data
- Processes for key management and permission granting can be re-used
- Top security due to support of various hardware security modules (HSM)
- No downstream costs caused by encryption (reuse of key management)
- Compliance with security requirements and standards (PCI DSS)

### Implementation

The following FLAM5-based solutions are currently available:

- OpenPGP-compliant (RFC 4880) protection of files
  - PKCS#11 HSM or soft tokens
  - IBM CCA-based HSM (inclusive ICSF on z/OS)
  - PGP keyrings as software implementation

## 1.1. The use of FKM5 in FLAM

This chapter describes the usage of FKM5s developed by limes datentechnik. Calling custom developed FKM5 works in the same way, but mostly has another structure for the FKM5 parameter string.

To call a simple load module on ZOS, only the function name must be specified. To call a function of *libfkm5* on ZOS, the library name of the DLL and function name must be specified. On other platforms (Windos, UNIX), the default library name is *libfkm5* and the default function name depends on the context. For example, if you want to encrypt a PGP file, then PGPCCA on IBM platforms and PGPP11 on all other platforms is used.

### NOTE

In some environments, you have to escape quotation marks of the FKM5 parameter string with backslashes: \".

## 1.2. PGPRING *SWE*

## SYNOPSIS

```
HELP:   FKM5 key ring parameter for PGP
PATH:   limes.
TYPE:   OBJECT
SYNTAX: :>  SWE(PUBSTORE='str',SECSTORE='str',PASSWORD='bin',NEWPASSWORD='bin')
```

## DESCRIPTION

The FKM5 function PGPRING is an FKM5 which uses conventional OpenPGP keyrings to implement RFC4880 aka. OpenPGP. The FKM5 implements data key generation, import and export functions for session key exchange, signing and verification of hash values. Data encryption with a session key and hash calculations is performed by the PGP component itself.

The specification can be found at the URL below:

```
https://www.ietf.org/rfc/rfc4880.txt
```

The FKM5 parameter of PGPRING function is used to specify public and private keyring file and the passphrase to decrypt the private keyring.

Example: Encrypt OpenPGP file with libfkm5 PGPRING.

```
flcl conv
  read.file='~.TEST.FB'
  write.text(file='~/test.pgp.a64'
             encrypt.pgp(userid='receiver'
                         fkm5(lib ='libfkm5'
                              func='PGPRING'
                              para='pub=pubrng.pgp,sec=secrng.pgp,pass=1234'
                             )
                        )
            )
```

All active operations based on the user ID (signing/encryption) automatically select the youngest of currently valid keys. If there is more than one matching key (multiple keys with same user ID and creation date), the key with the shortest validity period is chosen. This allows to pre-generate keys (see parameter ACTIVATION) for regular key exchange and also the ad-hoc exchange of keys if a key is compromised.

User IDs are usually of the form "user name (comment) <email>", where comment and email are optional. The desired key is selected by specifying "user name" or "email" or the whole user ID. Usage of wildcards in the user ID specification is also possible.

Example: Delete all OpenPGP keys on PKCS#11 ending with "@flam.de"

```
$ flcl key "del.pgp(user='*@flam.de'
                    fkm5(libr='libfkm5' func='PGPRING'
                    para='pub=puprng.pgp,sec=secrng.pgp,pass=123456'))"
```

You can use the KEY LIST.PGP() function to show the current status of all your available PGP keys.

Example: List all OpenPGP keys on PKCS#11

```
$ flcl key "list.pgp(fkm5(lib ='libfkm5' func='PGPRING'
                     para='pub=puprng.pgp,sec=secrng.pgp,pass=123456'))"
```

### ARGUMENTS

- STRING: PUBSTORE='str' - Key file or key ring with public keys (certificates)
- STRING: SECSTORE='str' - Key file or key ring with private keys
- STRING: PASSWORD='bin' - Password for encrypted private keys [optional (not required for public keys)]
- STRING: NEWPASSWORD='bin' - New password to re-encrypt private keys [only in key import/delete functions]

## 1.3. PGPP11 *PKCS11*

### SYNOPSIS

```
HELP:   FKM5 PKCS#11 parameter for PGP
PATH:   limes.
TYPE:   OBJECT
SYNTAX: :>  PKCS11(LIBRARY='str',SLOT=num,PIN='str')
```

### DESCRIPTION

The FKM5 function PGPP11, is an FKM5 which uses PKCS#11 to implement RFC4880 known as OpenPGP. The FKM5 implements the data key generation, import and export functions for session key exchange, signing and verification of hash values. The mass data encryption with the session key and hash calculations is done by the PGP component itself. Only the private key operations are protected by the crypto token interface conform hardware security module or soft token.

For session key exchange and signing RSA with 1024, 2048 and 4096 bit long keys is currently supported. The supported symmetric ciphers, hash and compression algorithms for PGP are defined by the PGP component using this key management extension.

This FKM5 function can be used with any kind of PKCS#11 library on any supported platform. The specification was designed for PCI DSS compliant ordering of credit cards and to exchange other card holder data in a secure and PCI DSS compliant manner.

The specification can be found here:

```
https://www.ietf.org/rfc/rfc4880.txt
```

To load the required PKCS#11 functions, the library (SO/DLL) must be specified. The default library is *libcryptoki*. The directory for this DLL/SO must be set in the library path environment variable or you

must specify an absolute path and library name.

Additionally, you must define the slot number (default is 0) and the corresponding PIN to enable access to the soft token or hardware security module (HSM).

To manage the PGP meta data, the PKCS#11 token must support data objects.

Example: Encrypt OpenPGP file, with libfkm5 PGPP11.

```
flcl conv
  read.file='~.TEST.FB'
  write.text(file='~/test.pgp.a64'
             encrypt.pgp(userid='receiver'
                         fkm5(lib ='libfkm5'
                              func='PGPP11'
                              para='lib=libcryptoki.so,slot=1,pin=1234'
                             )
                        )
            )
```

All active operations based on the user ID (signing/encryption) automatically select the youngest of currently valid keys. If there is more than one matching key (multiple keys with same user ID and creation date), the key with the shortest validity period is chosen. This allows to pre-generate keys (see parameter ACTIVATION) for regular key exchange and also the ad-hoc exchange of keys if a key is compromised.

On key deletion, the user ID based key labels for the primary, subkeys and also user plus key ID based data object labels are removed from the key store. You can specify a timestamp to delete an older or younger key instead of the currently active key.

At key export you can also specify the point in time to export a newer or older key. If no timestamp is given, the current time is used to choose the youngest of the active keys.

If no key ID is available during a read operation, you can provide the recipient user ID and the time stamp to choose the correct key (hidden recipient support).

With key list, all matching key labels are listed. For all PGP keys the activation and expiration date is provided together with user ID, key ID, key type (TOP/SUB-key) and public key algorithm.

You can use the KEY LIST.PGP() function to show the current status of all your available PGP keys.

Example: List all OpenPGP keys on PKCS#11

```
flcl key list.pgp(fkm5(lib ='libfkm5' func='PGPP11'
                       para='lib=libcryptoki.so,slot=1,pin=1234'
                      )
                 )
```

If you generate or import PGP keys with the same creation date and more than one key is active, then the system chooses the key with the shorter valid period.

The time bar for keys allows complete live cycle management for keys which is very simple to use. Other PGP implementations may not accept pre-generated keys (creation time in the future). In this

case don't, use a future point in time at key export.

### ARGUMENTS

- STRING: LIBRARY='str' - PKCS#11 library name (CRYPTOKI DLL/SO) [DEFAULT]
- NUMBER: SLOT=num - Slot number of PKCS#11 token [0]
- STRING: PIN='str' - PKCS#11 pin for authentication

## 1.4. PGPCCA *CCA*

### SYNOPSIS

```
HELP:   FKM5 CCA/ICSF parameter for PGP
PATH:   limes.
TYPE:   OBJECT
SYNTAX: :>
CCA(LIBRARY='str',USERID='str',PASSWORD='str',KEYSTORE='str',CLEARKEY,KEYTEMPLATE='str
')
```

### DESCRIPTION

The FKM5 function PGPCCA is an FKM5 which uses IBM CCA (Common Cryptographic Architecture (ICSF on ZOS)) to implement RFC 4880 known as OpenPGP. The FKM5 implements the data key generation, importing and exporting functions for session key exchange, signing and verification of hash values. The mass data encryption with the session key and hash calculations is done by the PGP component itself. Only the private key operations are protected by the IBM 47xx/CEXx hardware security module.

For session key exchange and signing RSA with 1024, 2048 and 4096 bit long keys is currently supported. The supported symmetric, hash and compression algorithms for PGP are defined by the PGP component using this key management extension.

This FKM5 function can be used with ICSF on ZOS (CEXx) or IBM47xx cryptocards on Windows or Unix platforms. The specification was designed for PCI DSS compliant ordering of credit cards and to exchange other card holder data in a secure and PCI DSS conform manner.

The OpenPGP specification can be found at the URL below:

```
https://www.ietf.org/rfc/rfc4880.txt
```

For ICSF, the library to find the callable services must not be specified because all functions are simple load modules in the dataset *CSF.SCSFMOD0*. These load modules are fetched by *libfkm5*. If you would specify a library name, *libfkm5* would try to load a ZOS-DLL, which would not work. To fetch the ICSF service routines, the *CSF.SCSFMOD0* library must be in the STEPLIB concatenation for the program.

The default library name for SAPI on windows is *csuncca* and on UNIX systems *libcsulcca*. The directory for this DLL/SO must be defined in the library path environment variable.

On ICSF-based CCA systems (z/OS), authentication against the CCA HSM is not possible. For such an

environment, please don't use the userid and passphrase login. On other systems, the userid and passphrase are optional. If you don't provide them, the default role is used.

To manage the PGP meta data in the PKDS, the key storage file can be specified. On ICSF, this could be a used (initialized) PKDS or a normal VSAM dataset allocated like a PKDS and allocated and initialized by job FLPGPINI taken from FLAM JOBLIB. If a real initialized PKDS is specified, all PGP related stuff is managed with one key data set (the current ICSF PKDS). This solution requires to store each key twice. Once under the user ID with all the PGP attributes (key version, creation date, valid days, ...) and under the PGP key id. If an additional not ICSF but FLPGPINI initialized VSAM dataset is given, then the PGP meta data is stored in this additional dataset and the PKA key tokens are stored only under the PGP key ID in the actual PKDS. Until z/OSv2.2 (FMID HCR77B0), the single PKDS solution requires unqualified VSAM access (LOCATE) to the PKDS to find the required key label. This direct VSAM access is only possible if the dataset is not locked. In an environment where the PKDS is written too frequently by other applications, this lock prevents writing OpenPGP files. Normally, this is not the case, but to offer a usable workaround for this situation, we have implemented the solution to work with the real ICSF PKDS and an additional VSAM dataset to manage the PGP meta data. With z/OSv2.2 a new ICSF (HCR77B0) verb CSFKDSL is available to search keys in the key store. If this verb is available, the PKDS is only required to list the content of a certain key data set. Since FLAMv5.1.13, a PGP trust store (simple sequential dataset or USS file) can also be used to prevent the lock issue and to use long user IDs containing email addresses. With a additional PGP trust store there are no limitations compared to an OpenPGP key ring solution. You can start your migration project by using the original PGP key rings on z/OS and later you can import your secret key file containing the private keys to ICSF without any differences to the software solution except there is no password for the private keys required anymore. If you start to work with the FLAM OpenPGP support, you must decide which solution is better for your environment.

When reading, the key ID is usually used for fully qualified access to the corresponding key over ICSF. Without the CSFKDSL (HCR77B0) service, the PKDS or trust store name must be provided to write a PGP file.

The job FLPGPINI allocates a VSAM-KSDS like the ICSF-PKDS but writes a different unique initialization record to this dataset. Only if this special initialization record is found, the dataset will be written by FLAM. This prevents writing to a real ICSF-PKDS by mistake.

The solution with two key stores does not support the RECOPY option of the key generation function, because there is no key token stored under the PGP user ID. If you export your public key to a PGP key file and you import this key file with OVERWRITE, then you lose your private key, because only the public key is in the key file and will overwrite your key pair in the key store. With the one-key-store solution, you can RECOPY the key pair from the user ID under the corresponding key ID to fix this issue.

The solution with the real PKDS as the only used key store cannot store the preferences defined in an imported PGP certificate. In this case, the preferred algorithms, procedures and so on are set by the PGP component depending on the available hardware acceleration capabilities (CPACF) during export. With the two PKDS files or the trust store, these attributes are stored in the FLAM owned key store file and can be restored on key export. With the trust store there is no restriction on user IDs anymore. If you use one of the PKDS solutions, the user ID is limited to two qualifiers with 8 characters, each separated by a period. You can define the new short user id at import and you can set the original one at export.

On platforms using the CCA support program (not ICSF) the callable service CSNDKRL is used to determine the complete key label based on the user ID. With this service, the lock issue will not happen and the solution with the second PKDS is not required and not supported. The KEYSTORE parameter is only required to list the content of a certain key file from disk or to use the trust store solution. Also, the key list function uses CSNBKRL or CSFKDSL if no keystore filename is provided.

A CCA implementation in general is limited to the capabilities of ICSF/CCA concerning the public key operations. Currently this limits to RSA. In the future, if it is specified for OpenPGP and implemented by the IBM cryptocard, we can support ECC but ElGamal, DSA and other asymmetric algorithms

cannot be supported with this HSM.

To implement a CCA solution working only with the PKDS, the key label templates (!FLAMPGP.%u.%t) can be changed, but the type (%t) must separated with a dot always at the end of the template. If the key type in any other place the parsing of the PGP attributes (creation date, valid days, key version and usage) will fail or the right keys are not found. This FKM5 will issue an error if *.%t* is not at the end of the key templates.

Since version 5.1.16, the replacement characters for key label templates are no longer case sensitive. This simplifies usage in JCL with CAPS ON.

For the operational PGP stuff the CCA verbs below are required:

- CSNDPKX - Public key extract
- CSNDRNGL - Random number generator in variable length
- CSNDPKE - PKA encrypt
- CSNDPKD - PKA decrypt
- CSNDDSG - Digital signature generate
- CSNDDSV - Digital signature verify
- CSNDKRL - Key record list (not required/available on ICSF)
- CSFKDSL - Key dataset list (requires ICSF FMID HCR77B0 (z/OSv2.2))
- CSUALCT - Only if user ID password login used (not required/available on ICSF)

For the PGP key management the additional CCA verbs below are required:

- CSNDKRR - PKA key record read
- CSNDKRC - PKA key record create
- CSNDKRW - PKA key record write
- CSNDKRD - PKA key record delete
- CSNDPKB - PKA key token build
- CSNDPKI - PKA key import
- CSNDPKG - PKA key generate

To separate the PGP key management from the operational stuff, we recommend to define different roles for it. The key management functions do not handle secret key material in clear form. This means that single control for this operations are enough. With the trust store solution the key management for FLAM-PGP can be supported by DKMS/EKMF.

The PGP user IDs are limited to 2 key label qualifiers if the PKDS-only solution is used. Please rename the original PGP user ID during import of key files from external partners. During export, you can provide a new user ID for the PGP certificate written to the PGP file. This mechanism allows to remove the email address at import and add an email address at export. With the new trust store solution, this shorting is not required and you can access the key by email or user name.

The CCA implementation limits the CCA usage for a PGP subkey to session key decryption. The primary key without a subkey can be used for key exchange and signing. If a subkey is available, the usage of the PGP primary key is limited to signature generation.

The internal key type (used for *%t* in the key label template) for user id based key labels needs two key label qualifiers:

```
 xnvdddd.cccccccc
 ||||||| ++++++++ PGP creation date in Base 32 (40 bit in extended HEX (seconds since
1970))
 |||++++--------- PGP valid days in Base16 (16 bit in HEX (days after creation))
 ||+------------- PGP key version (3 or 4)
 |+-------------- PGP subkey number (0-F (4 bit HEX, primary key = 0))
 +--------------- PGP primary key has T (TOP) or subkey has S
```

For file encryption a primary/top key and a subkey are required. This is the default at key generation. The number of subkeys at key import is limited to a maximum of 16.

At key export and import the CCA implementation supports version 3 and version 4 PGP keys. Generated keys are always version 4 keys.

Checking whether the key is not expired (valid days) occurs at signature generation or data key encryption (write). This is not controlled by the CCA-based HSM. During passive operations (signature verification or session key decryption (read)), the key expiration is not verified.

The creation date must be stored as part of the key type to recreate the PGP key id for version 4 keys. This complex key type specification for user ID based key labels allows to manage all the PGP stuff within the PKDS. No other key stores are required to exchange PGP files with FLAM.

The key type for key id based key labels is the constant *PGPKEYID*. Based on the standard key label template for PGP, one PGP key consists of four different labels which point to two different key values for the primary (top) and subkey.

```
Primary PGP key:
  User ID based label:  !FLAMPGP.userxxxx.idxxxxxx.T04dddd.cccccccc
  Key ID based label:   !FLAMPGP.keyxxxxx.idxxxxxx.PGPKEYID
Sub PGP key:
  User ID based label:  !FLAMPGP.userxxxx.idxxxxxx.S04dddd.cccccccc
  Key ID based label:   !FLAMPGP.keyyyyyy.idyyyyyy.PGPKEYID
```

All active operations based on the user ID (signing/encryption) automatically select the youngest of currently valid keys. If there is more than one matching key (multiple keys with same user ID and creation date), the key with the shortest validity period is chosen. This allows to pre-generate keys (see parameter ACTIVATION) for regular key exchange and also the ad-hoc exchange of keys if a key is compromised.

On key deletion, the user ID based and key ID based key labels for the primary and all subkeys are removed from the key store. You can specify a timestamp to delete only the corresponding youngest key.

At key export you can also define the point in time to export only one key set. If no timestamp is given, all keys for the user are exported.

If no key ID is available when reading, you can provide the recipient's user ID to choose the correct key (hidden recipient support).

With key list, all matching key labels from the given or current PKDS are listed. For this list, the provided VSAM data set name is used. If this an initialized ICSF PKDS, then the user ID and key ID based labels are listed. If the solution with a separate PKDS or trust store for the PGP meta data is used, only the user ID based labels are listed together with the corresponding key ID. For all PGP keys,

the activation and expiration date is provided together with the status information below:

- ACT - for an active (valid) PGP key
- WRN - for a PGP key which will expire in the next days (warning)
- EXP - for an expired PGP key (not usable for encryption and signing)
- INA - for a inactive PGP key (activation date is in the future)
- KID - for a PGP key ID based key
- CCA - for all other keys

You can use the KEY LIST.PGP() function to show the current status of all your available PGP keys.

If you generate or import PGP keys with the same creation date and more than one key is active, then the system selects the key with the shorter valid period. If the valid period is also identical, then the version 4 keys are preferred to version 3 keys. If also the version is identical then the first of that keys is used.

The time bar for keys allows complete life-cycle management for keys which is very simple to use. Other PGP implementations might not accept pre-generated keys (PGP creation in the future). In this case, don't use a future point in time at key export.

Example: Encrypt OpenPGP file with libfkm5 PGPCCA on MAINFRAME.

```
flcl conv
  read.file='~.TEST.FB'
  write.text(file='~/test.pgp.a64'
             encrypt.pgp(userid='receiver'
                         fkm5(lib ='libfkm5'
                              func='PGPCCA'
                              para='keystore=CSF.CSFPKDS'
                             )
                        )
            )
```

On ICSF based systems, the library and function names are not required and the keystore name is only required for writing if CSFKDSL (HCR77B0) is not available and the solution with the 2 PKDS or PKDS with trust store are used.

Example: Encrypt OpenPGP file with libfkm5 PGPCCA on Linux.

```
flcl conv
  read.file='~/test.txt'
  write.text(file='~/test.pgp.a64'
             encrypt.pgp(userid='receiver'
                         fkm5(lib ='libfkm5'
                              func='PGPCCA'
                              para='lib=/usr/lib/libcsulcca.so'
                             )
                        )
            )
```

On CCA based UNIX systems only the library name is required if the library has another name or if the library is not in the library path.

## ARGUMENTS

- STRING: LIBRARY='str' - CCA library name (SAPI DLL/SO or empty for ICSF) [DEFAULT]
- STRING: USERID='str' - User ID for authentication [optional (not required for ICSF)]
- STRING: PASSWORD='str' - Password for authentication [optional (not required for ICSF)]
- STRING: KEYSTORE='str' - Used CCA/ICSF key store (PKDS/CKDS) [optional (PKDS required for PGP-KM)]
- SWITCH: CLEARKEY - Activate use of clear private keys at generate and import [OFF]
- STRING: KEYTEMPLATE='str' - Used CCA key label template [!FLAMPGP.%u.%t]

# INDEX

# COLOPHON

limes datentechnik(R) gmbh
Louisenstrasse 101
D-61348 Bad Homburg v.d.H.
phone: +49(0)6172-5919-0
fax:   +49(0)6172-5919-39
mail:  info@flam.de
web:   www.flam.de or www.limes.de
Amtsgericht: Bad Homburg vor der Hoehe HRB 3288 (gegr. 1985)
Geschaeftsfuehrer: Diplom-Mathematiker Heinz-Ulrich Wiebach
limes datentechnik(R): efficiency at the limit of possibility.